Virtualisation

Project 1

Alex Harper

Contents

List of Figures

1 Introduction

This project has been my first time using Visual Studio and git on Windows. While I do not really like either things, they do work if you use them correctly. In this paper I go over what I did to get ready for working with Windows for programming as well as some specific programming bits.

The programming I show here is mostly for me to play with Visual Studio to work out what quirks it has. I took that opportunity to get Goole Test setup in a project and do some basic tests, which was a bit of a learning curve for me. For the tests, I was pushing how far I could get bad code to still work correctly by doing obviously wrong things. I also give some basic code examples of making classes, because that is what the book goes over.

2 Visual Studio

Since I am always the odd man out by using linux, I have to cave and meet the enviroment that my group partners are going to be using. This means Windows and Visual Studio. I have touched Visual Studio before in the form of the reskined Atmel Studio to do micro controller programming, but that was fairly different than the original.

Installing the IDE was simple with downloading the installer and running it. It gave some options and I made sure to get the C++ compiler installed.

After getting installed, it was time to mess around with it and see what it is like. I made a new "solution" and made sure the compiler worked, and that things operated for the basics. It is fairly similar to what I have used before, just different keyboard bindings.

After messing around a few minutes, I made a list of things I don't like about the program, but in general are not very major.

- Missing Features that I am used to having
  - When the variable is a pointer to an object, pressing the '.' button does not automatically translate it into '->'

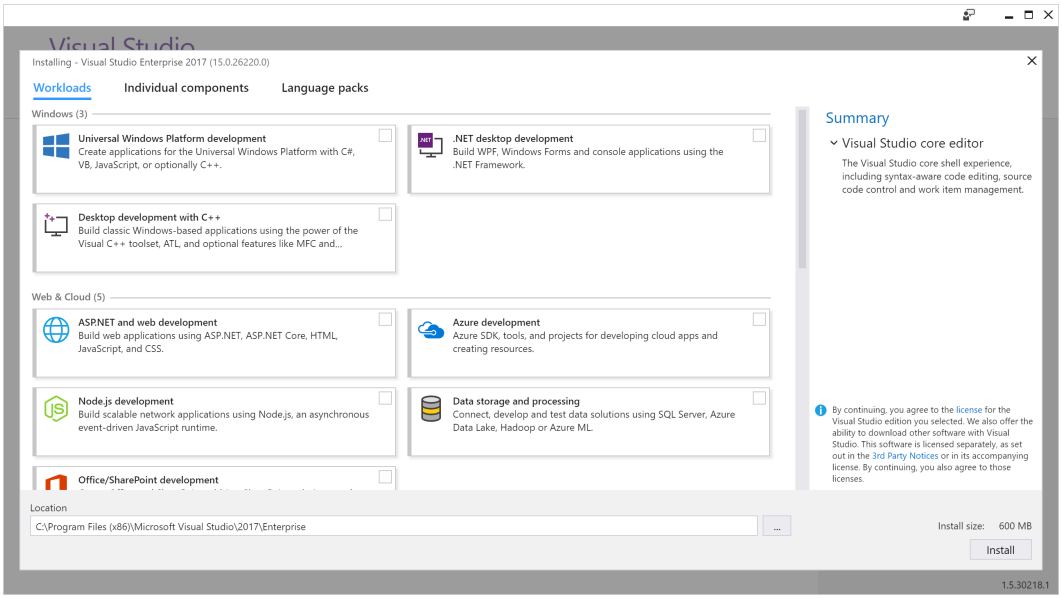


Figure 1: Window of Installer to Select extra Components

  - The suggestions pane while typing does not automaticly come back when I accidentally dismiss it (but ctrl+space brings it up)
  - The suggestions pane while typing does not select the first entry in the box by default and waits for me to press the down button. This messes me up a lot with my muscle memory
  - The block comment shortcuts are stupid. It should be a toggle of commenting the block and not make the person have to remember two different buttons.
  - The "toggle header/source file" shortcuts don't even work

- When asking it to build or start a test, it takes a second or two to even start the process. There is no reason it should take that long
- It broke on me once, simply not building. A simple restart of the program fixed it, but why did it decide to stop working?
- There seems to be no way to select a location in a repository to put a solution made in visual studio, making me have to externally manage it. Not a huge deal

3 Google Test

For the testing framework, our group has decided to use Google Test. It is a simple framework that lets you simply define a group of code and explicitly set test conditions. The framework handles running the tests and trys to gracefully handle any exceptions in the tests (and marks those as failures). To install the framework, it was simple with the built in tool that manages extra plugin stuff for Visual Studio. In the start menu search for "visual studio installer" and then click the buttons shown in fig2 and fig3.

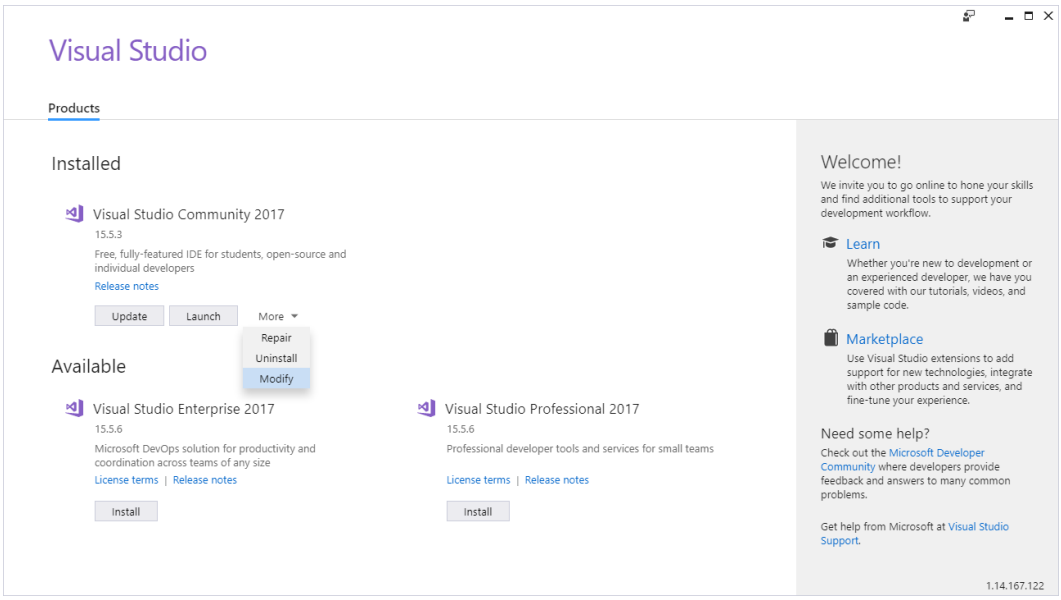


Figure 2: Button To Press In The "Installer" To Add Components To Visual Studio

After getting it installed, it is time to make a new project to use google test. This is where I got confused and spen about an hour going in circles. Visual Studio has two different things of a "solution" and a "project". Only a single solution can be open at a time, but it can contain several projects. I had thought that they were the same thing and as I went to make a new project, instead I made a new solution. When you go to "File - New - Project" it defaults to making a new solution at the same time and not putting it in the current solution which already had my code in it. There is a dropbox for making it add to the current solution, but instead I will continue to opt for what is fig4.

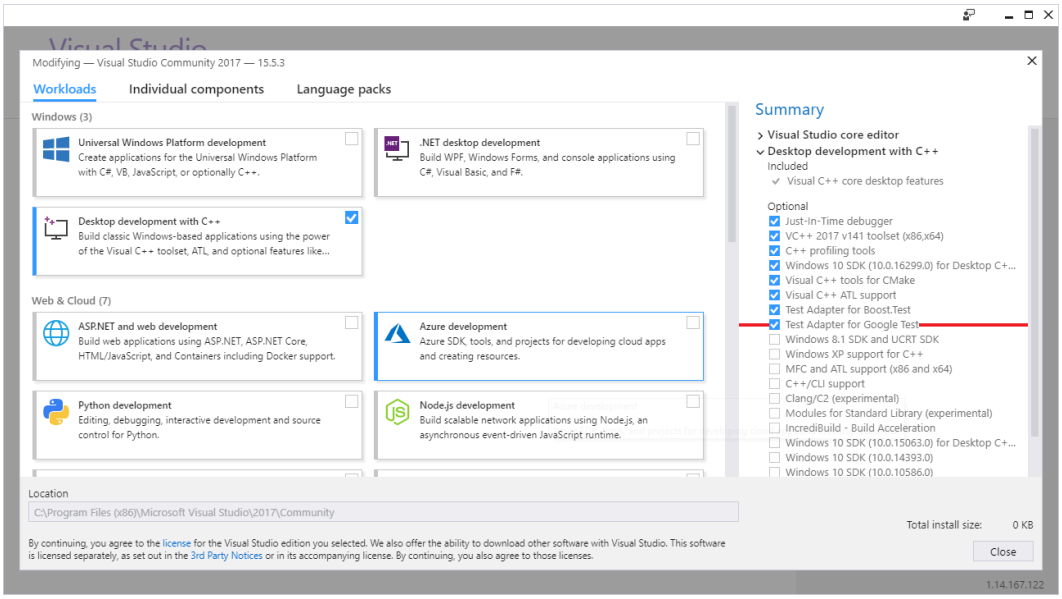


Figure 3: Checkbox Location For Google Test - Highlighted In Red

And now that the framework is installed, lets make some tests. At first I decided to only do super basics, but it devolved into a short exploration of abusing the memory layout of objects and see what technically works and didn't break things. In fig5 is an example test case that shows the very basic usage. In fig6 shows the test explorer pane in visual studio.

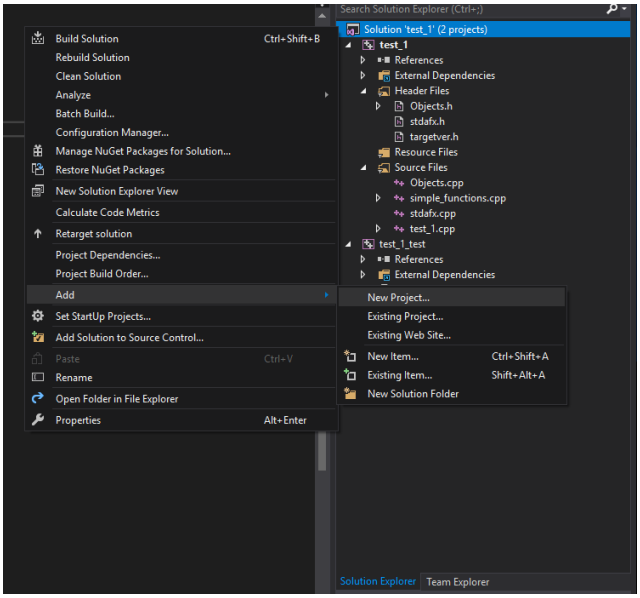
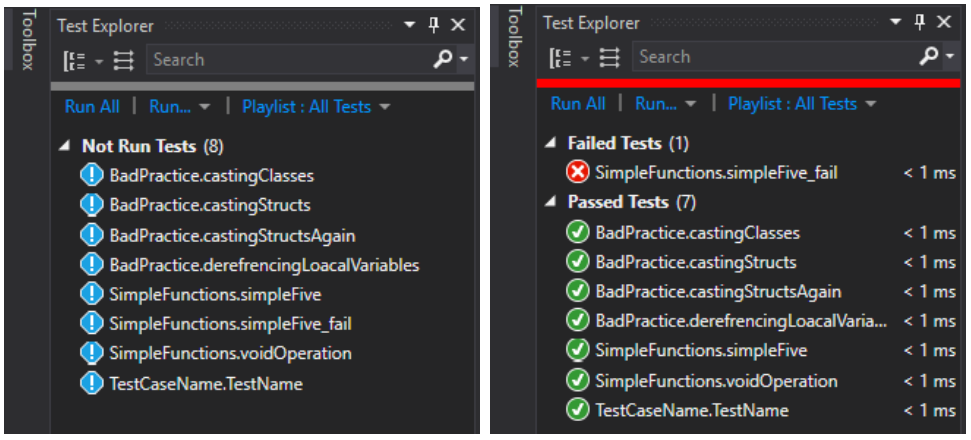


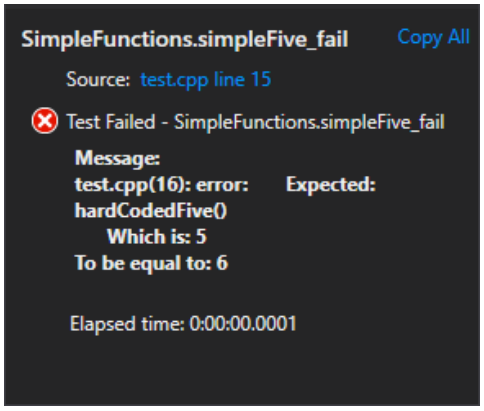
Figure 4: How I Will Add Projects To My Current Solution From Now On

```
1 TEST(TestGroupName, TestName) {
2     EXPECT_EQ(1, 1);
3     EXPECT_TRUE(true);
4 }
```

Figure 5: Example Test Case Using Google Test



(a) Tests Before Running Them (b) Tests After Running Them

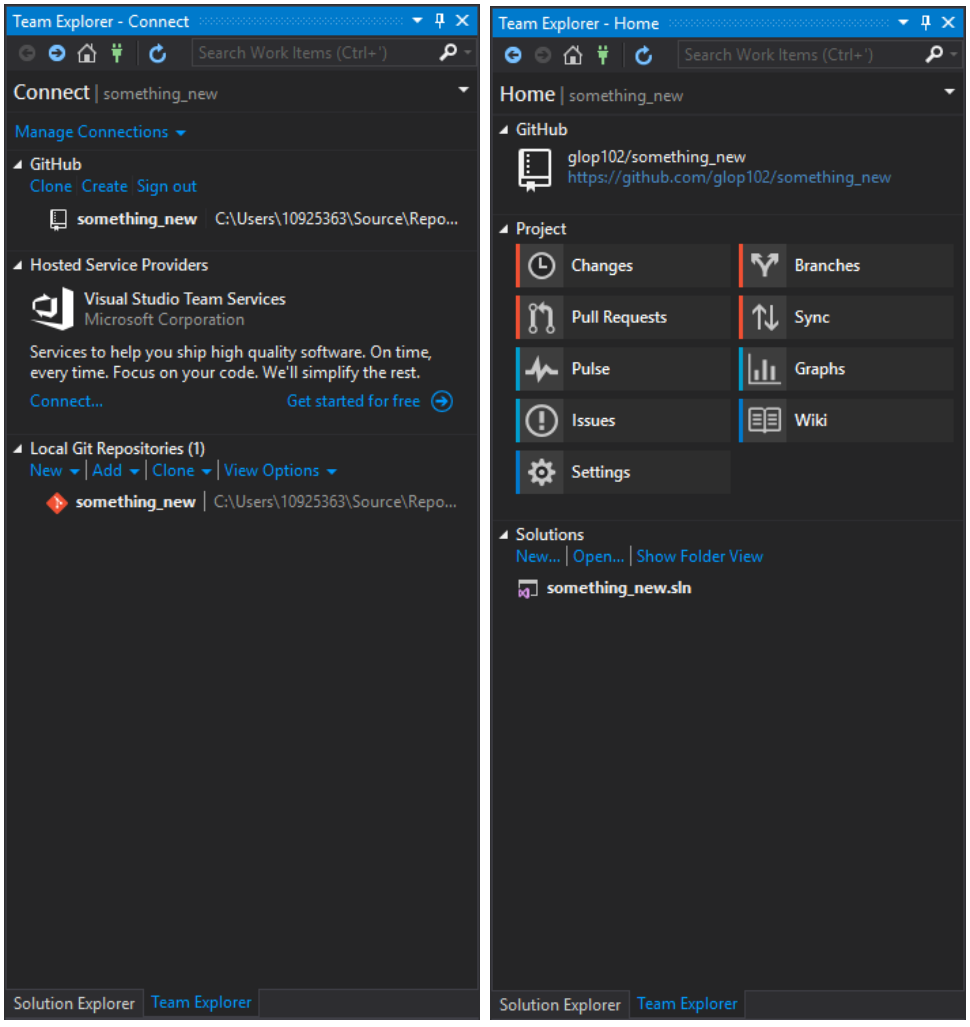


(c) Summary of a Failed Test

Figure 6: Test Explorer Pane In Visual Studio

4 Git on Windows

This is where things took a large departure from what I was expecting. I am very familure using git with a command line and I like the simplicity of it. The extension in visual studio is something that I am not used to, as it feels opaque to me. Really the operation of it is simple and the same as normal git, but I still like my command line. In fig7 is a picture of what the add-on looks like.



(a) Team Explorer With Github (b) Local Repo with Various Buttons

Figure 7: Team Explorer Pane In Visual Studio

The team explorer pane is a builtin part of the program already, but is limited to only working with microsoft services. The github extension adds the extra field of letting github be the back end that code is stored on. As you can see in fig7b, it has quick access buttons for the typical actions. The "changes" button is where you make commits to the current branch. The "sync" button is where you push or pull code. The "branches" button is where you do merges and such.

After messing with it, I suggest starting by making the new repo with github first. Place the repo where you want before making the new "solution" in VS. When you make the new "solution", change the location to the repo folder that was just made. Now when you go to the changes button with the plugin, it will let you add a commit message and push the files up. Note: When you make the repo, make sure to have a .gitignore file get made to keep all the trash that visual studio puts in from making your repo get huge in size.

While this seems kindof handy, I still prefer using the command line directly. I am used to a workflow on the command line under linux. The idea of an IDE is neat, but I typically only use it for a handy text editor that gives suggestions as I type. It will be an interesting experiance as I work with my team on this.

5 Playing With Classes

As the section fo the book is about how to make classes with various restrictions and features, I suppose I should mention some things about that myself. I simply added this code into my test project for Google Test.

The book has many pages about specifics of what to do for exact situations. I instead am just going to give a short list of things here that sumarize the important bits.

In fig8 is the most simple example subclass you can do in c++. The test on the right passes. The first two conditional checks show that things make sense with each returning their own proper names. The last conditional check has the subclass casted as a base class, and so it returns the name of the base class.

```
class Base1 {
public:
    string name();
};
class Sub1:public Base1 {
public:
    string name();
};
```

```
TEST(ClassTest,t1){
    Base1 *a, *bb;
    Sub1 *b;
    a = new Base1;
    b = new Sub1;

    string name = a->name();
    EXPECT_EQ(name, "Base1");

    name = b->name();
    EXPECT_EQ(name, "Sub1");

    bb = (Base1*)b;
    name = bb->name();
    EXPECT_EQ(name, "Base1");

    delete a;
    delete b;
}
```

Figure 8: Simple Subclass

In fig9 is trying to change the last test case. The way to do this is to declare the name() method as virtual in the base to allow sub classes to override it. As before, the first two conditional checks have them returning their own proper names. The last conditional check has the subclass casted as a base class, but this time it still returns the name of the subclass. This shows how virtual functions can allow extensions to an object without changing code that uses the objects.

```
class Base2 {
public:
    virtual string name();
};
class Sub2 :public Base2 {
public:
    string name();
};
```

```
TEST(ClassTest, t2) {
    Base2 *a, *bb;
    Sub2 *b;
    a = new Base2;
    b = new Sub2;

    string name = a->name();
    EXPECT_EQ(name, "Base2");

    name = b->name();
    EXPECT_EQ(name, "Sub2");

    bb = (Base2*)b;
    name = bb->name();
    EXPECT_EQ(name, "Sub2");

    delete a;
    delete b;
}
```

Figure 9: Virtual Methods for Subclasses

In fig10 shows the last major topic in objects. A static variable is tangentially related to global variables, in that there is only one that everyone shares. When you declare a thing static in an object, it is going to be the single thing used for all objects of that type. Say you have a static int and you make 100 objects, all 100 objects would use the exact same int, and not their own copies of it. Same idea with inheriting a static method, it will be the only method that everyone uses with not even a chance to override it. That is why in this example the "Sub3" class does not even make it's own. It is forced to use the base class and not allowed to overwrite it.

```
class Base3 {
public:
    static string name();
};
class Sub3 :public Base3 {};
```

```
TEST(ClassTest, t3) {
    Base3 *a;
    Sub3 *b;
    a = new Base3;
    b = new Sub3;

    string name = a->name();
    EXPECT_EQ(name, "Global");

    name = b->name();
    EXPECT_EQ(name, "Global");

    delete a;
    delete b;
}
```

Figure 10: Static Methods for Subclasses

6 Abusing Memory Layout

And now the fun part I was doing before making the example classes above. I like sometimes just trying to take things too far while keeping them from breaking. This is a good way to make sure you understand what is really going on in the background. While I do these things here, it is me just having fun. They are all really terrible ideas and will eventually destroy your code if you do them.

So, first was me making sure I can do things with structs that I had done before. Structs are great because they are just simple laied out containers of a certain number of bytes. This lets you abuse the memory layout on a system to do more stupid things than normal. In short you are able to pass one type of struct as if it were a different kind and things still work. The first example shows two structs with the same layout but different names. That is easy to understand how it works, since really there is no difference between them. fig11

```
struct S1 {
    int a, b;
};
struct S2 {
    int c, d;
};
```

```
TEST(BadPractice, castingStructs) {
    struct S1 *s = new struct S1;
    changeS1(s);
    EXPECT_EQ(s->a, 100);
    EXPECT_EQ(s->b, 50);
    changeS2((struct S2*)s);
    EXPECT_EQ(s->a, 4);
    EXPECT_EQ(s->b, 5);
    delete s;
}
```

Figure 11: Simple Struct Casting

But this is where it bends your mind, a struct of a different form. The second struct is made out of more, smaller variables. Since we are familure with memory being a linear arrangment of bytes, we can think about how this could go. An int is by default 4 bytes, and an char is by default 1 byte. That means the first struct is 8 bytes long, while the second struct is 4 bytes long. So, if you can interpret the test, it does something as it casts the first struct to be of the second type. The function “changeUsingBytes” is a hint to it takes the number you pass in, breaks it into bytes, and then assigns the bytes to the

char variables. Since the test passes, you can assume it works for setting the number to variable 'a' even though we cast it. fig12

```
struct S1 {
    int a, b;
};
struct S3 {
    char first;
    unsigned char second;
    unsigned char third;
    unsigned char fourth;
};
```

```
TEST(BadPractice, castingStructsAgain) {
    struct S1 *s = new struct S1;
    changeS1(s);
    EXPECT_EQ(s->a, 100);
    EXPECT_EQ(s->b, 50);
    changeUsingBytes((struct S3*)s,1234567);
    EXPECT_EQ(s->a, 1234567);
    EXPECT_EQ(s->b, 50);
    changeUsingBytes((struct S3*)s, -1234567);
    EXPECT_EQ(s->a, -1234567);
    delete s;
}
```

Figure 12: Simple Struct Casting

And the last thing to try out is doing similar things with classes. Structs are a easy to understand thing, just a number of bytes put in memory. Classes though, they have extra bits, like functions and such. I decided to test how far it could go before running into problems. The answer, about as far as structs in my basic testing. While I will not put a code excerpt here, I will give a quick list of what works.

- Changing variable names
- Changing function names
- Adding more functions
- Casting it as a smaller object (fewer bytes in it)

But the thing that did not work was casting as a larger object. The problem only is that it when it tries to access the bytes at the end of the object, well, they were never allocated for your smaller object. This lead to my code having the program either endlessly sitting idle or the debugger finding heap corruption and stopping it.

A test.cpp - has the test cases in it

```
#include "pch.h"
#include <string>
#include "../test_1/simple_functions.cpp"
#include "../test_1/Objects.h"

using std::string;

TEST(TestCaseName, TestName) {
  EXPECT_EQ(1, 1);
  EXPECT_TRUE(true);
}

TEST(SimpleFunctions, voidOperation) {
    TestFunction();
}
TEST(SimpleFunctions, simpleFive) {
    ASSERT_EQ( hardCodedFive() ,5);
}
TEST(SimpleFunctions, simpleFive_fail) {
    EXPECT_EQ(hardCodedFive(), 6);
}

//Abusing that structs always do memory layout the same way
TEST(BadPractice, castingStructs) {
    struct S1 *s = new struct S1;
    changeS1(s);
    EXPECT_EQ(s->a, 100);
    EXPECT_EQ(s->b, 50);
    changeS2((struct S2*)s);
    EXPECT_EQ(s->a, 4);
    EXPECT_EQ(s->b, 5);
    delete s;
}
TEST(BadPractice, castingStructsAgain) {
    struct S1 *s = new struct S1;
    changeS1(s);
    EXPECT_EQ(s->a, 100);
```

```
38      EXPECT_EQ(s->b, 50);
39      changeUsingBytes((struct S3*)s,1234567);
40      EXPECT_EQ(s->a, 1234567);
41      EXPECT_EQ(s->b, 50);
42      changeUsingBytes((struct S3*)s, -1234567);
43      EXPECT_EQ(s->a, -1234567);
44      delete s;
45  }
46  TEST(BadPractice, derefrencingLoacalVariables) {
47      int *a = returningLocalVariable();
48      EXPECT_NE(*a,5); // so this didn't work as i expected, so i changed the test
49  }
50  TEST(BadPractice, castingClasses) {
51      //this works the same as the structs - if the sizes are the same, then every
52      Ex1 *a = new Ex1;
53      a->setValues();
54      EXPECT_EQ(a->a, 10);
55      EXPECT_EQ(a->b, 20);
56
57      ((Ex2*)a)->setValues();
58      EXPECT_EQ(a->a, 30);
59      EXPECT_EQ(a->b, 40);
60
61      ((Ex3*)a)->changeValuesToSomethingElse();
62      EXPECT_EQ(a->a, 50);
63      EXPECT_EQ(a->b, 60);
64
65      ((Ex4*)a)->changeValuesToSomethingElse();
66      EXPECT_EQ(a->a, 70);
67      EXPECT_EQ(a->b, 80);
68
69      ((Ex5*)a)->changeValuesToSomethingElse();
70      EXPECT_EQ(a->a, 90);
71      EXPECT_EQ(a->b,100);
72
73      ((Ex6*)a)->changeValuesToSomethingElse();
74      EXPECT_EQ(a->a, 110);
75      EXPECT_EQ(a->b, 120);
76
77      //this access an entierly new variable that our object doesn't have
```

```
78      //this is bound to crash programs when data changes over page lines or somet
79      ((Ex6*)a)->randomOtherFunction();
80      EXPECT_EQ(a->a, 700);
81      EXPECT_EQ(a->b, 800);
82
83      delete a;
84  }
85
86  //TEST(BadPractice, castingClasses_breaking) {
87  //      Ex1 *a = new Ex1;
88  //      Ex1 *b = new Ex1;
89
90      //this access an entierly new variable that our object doesn't have
91      //this time around, it seems to cause an infinate loop of some sort, and nev
92  //      ((Ex6*)a)->randomOtherFunction();
93  //      EXPECT_EQ(a->a, 700);
94  //      EXPECT_EQ(a->b, 800);
95  //      EXPECT_EQ(b->a, 900);
96
97  //      delete a;
98  //      delete b;
99      //even telling it to cancel the test does nothing to stop it
100     //the cpu is being used 0% but the program never actually exits
101     //i have to kill it under task manager
102     //the only operational difference is that i have defined a second object b
103     //when a cast as something else does an erronious write to a third variable,
104 //}
105
106 TEST(ClassTest,t1){
107     Base1 *a, *bb;
108     Sub1 *b;
109     a = new Base1;
110     b = new Sub1;
111
112     string name = a->name();
113     EXPECT_EQ(name, "Base1");
114
115     name = b->name();
116     EXPECT_EQ(name, "Sub1");
117
```

```
118     bb = (Base1*)b;
119     name = bb->name();
120     EXPECT_EQ(name, "Base1");
121 
122     delete a;
123     delete b;
124 }
125 
126 TEST(ClassTest, t2) {
127     Base2 *a, *bb;
128     Sub2 *b;
129     a = new Base2;
130     b = new Sub2;
131 
132     string name = a->name();
133     EXPECT_EQ(name, "Base2");
134 
135     name = b->name();
136     EXPECT_EQ(name, "Sub2");
137 
138     bb = (Base2*)b;
139     name = bb->name();
140     EXPECT_EQ(name, "Sub2");
141 
142     delete a;
143     delete b;
144 }
145 
146 TEST(ClassTest, t3) {
147     Base3 *a;
148     Sub3 *b;
149     a = new Base3;
150     b = new Sub3;
151 
152     string name = a->name();
153     EXPECT_EQ(name, "Global");
154 
155     name = b->name();
156     EXPECT_EQ(name, "Global");
157 
```

```
158     delete a;
159     delete b;
160 }
```

B simple_functions.cpp - some silly things I was testing on

```cpp
#pragma once
#include "stdafx.h"

void TestFunction() {
    //does nothing
}

int hardCodedFive() {
    return 5;
}

struct S1 {
    int a, b;
};
struct S2 {
    int c, d;
};
void changeS1(struct S1 *s) {
    s->a = 100;
    s->b = 50;
}
void changeS2(struct S2 *s) {
    s->c = 4;
    s->d = 5;
}

struct S3 {
    char first;
    unsigned char second;
    unsigned char third;
    unsigned char fourth;
};
void changeUsingBytes(struct S3 *s, int val) {
    s->first = (val >> 0 * 8) & 0xFF;
    s->second = (val >> 1 * 8) & 0xFF;
```

```
36      s->third = (val >> 2 * 8) & 0xFF;
37      s->fourth = (val >> 3 * 8) & 0xFF;
38  }
39
40  int* returningLocalVariable() {
41      int a = 5;
42      return &a;
43  }
44
45
46  class Ex1 {
47  public:
48      int a;
49      int b;
50      void setValues() { a = 10, b = 20; }
51  };
52  class Ex2 {
53  public:
54      int c;
55      int d;
56      void setValues() { c = 30, d = 40; }
57  };
58  class Ex3 {
59  public:
60      int e;
61      int f;
62      void changeValuesToSomethingElse() { e = 50, f = 60; }
63  };
64  class Ex4 {
65  public:
66      int g;
67      int h;
68      void changeValuesToSomethingElse() { g = 70, h = 80; }
69      void randomOtherFunction() { g = 700, h = 800; };
70  };
71  class Ex5 {
72  public:
73      int g;
74      int h;
75      void randomOtherFunction() { g = 700, h = 800; };
```

```
76      void changeValuesToSomethingElse() { g = 90, h = 100; }
77  };
78  class Ex6 {
79  public:
80      int g;
81      int h;
82      int q;
83      void randomOtherFunction() { g = 700, h = 800, q=900; };
84      void changeValuesToSomethingElse() { g = 110, h = 120; }
85  };
```

C Objects.h

```
#pragma once
#include "stdafx.h"

using std::string;

class Base1 {
public:
    string name();
};
class Sub1:public Base1 {
public:
    string name();
};

class Base2 {
public:
    virtual string name();
};
class Sub2 :public Base2 {
public:
    string name();
};

class Base3 {
public:
    static string name();
};
class Sub3 :public Base3 {};

class SummaryClass {
public:
    SummaryClass() {}
    virtual ~SummaryClass() {}

    void onlyLocallyOverridable() {}
    virtual void overridenIfASubclassWants() {}
    virtual void forcedToOverrideThis() = 0;
```

```
38
39      static void globalToAllObjectThatInherit() {}
40  };
```

D Objects.cpp

```cpp
#include "stdafx.h"
#include "Objects.h"

string Base1::name() {
    return "Base1";
}
string Sub1::name() {
    return "Sub1";
}

string Base2::name() {
    return "Base2";
}
string Sub2::name() {
    return "Sub2";
}
string Base3::name() {
    return "Global";
}
```